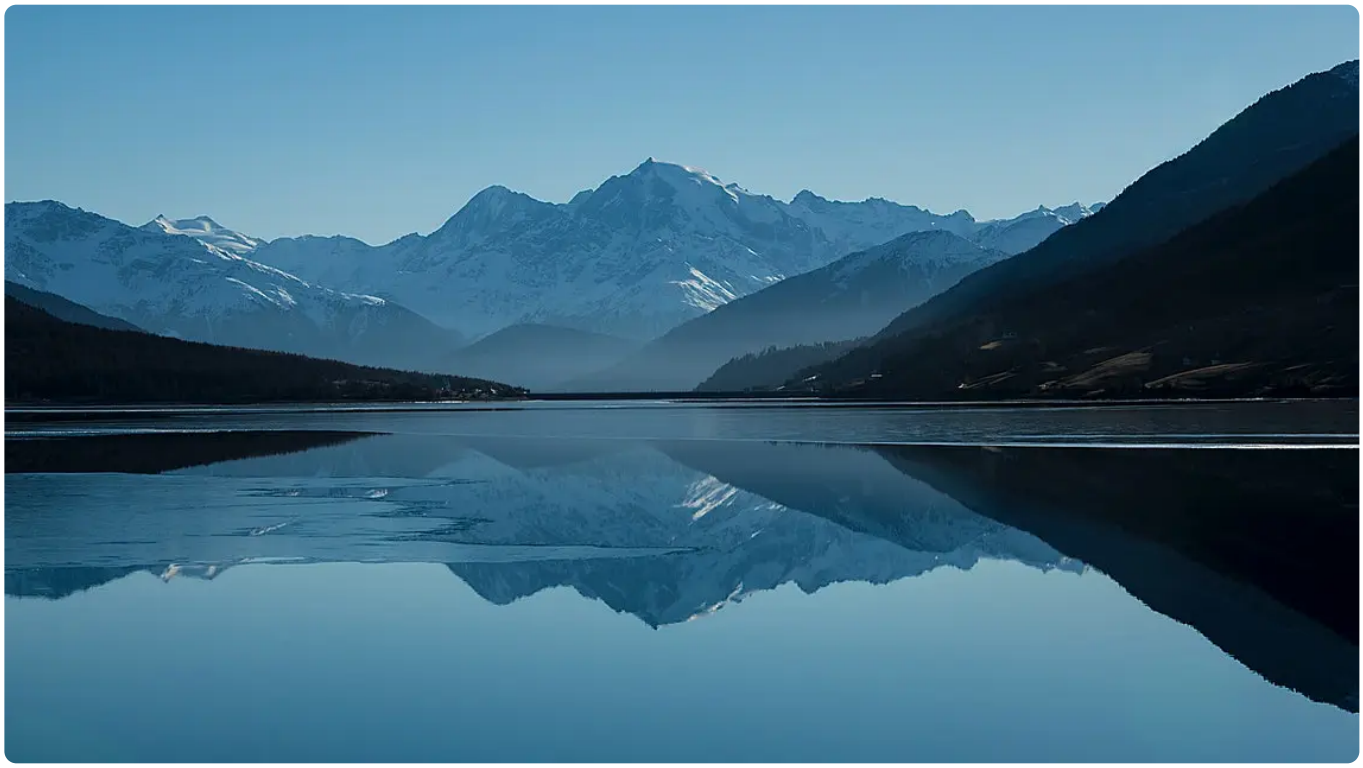
01 | JShell：怎么快速验证简单的小问题？

2021-11-15 范学雷

《深入剖析Java新特性》 课程介绍 >



讲述：范学雷
时长 14:03 大小 12.88M

你好，我是范学雷。今天，我们聊一聊 Java 的交互式编程环境，JShell。

JShell 这个特性，是在 JDK 9 正式发布的。从名字我们就能想到，JShell 是 Java 的脚本语言。一门编程语言，为什么还需要支持脚本语言呢？编程语言的脚本语言，会是什么样子的？它又能够给我们带来什么帮助呢？

让我们一起来一层一层地拆解这些问题，弄清楚 Java 语言的脚本工具是怎么帮助我们提高生产效率的。我们先从阅读案例开始。

阅读案例

学习编程语言的时候，我们可能都是从打印“Hello, world!”这个简单的例子开始的。一般来说，Java 语言的教科书也是这样的。今天，我们也从这个例子开始，温习一下 Java 语言第一课里面涉及的知识。

复制代码

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

好了，有了这段可以拷贝的代码，接下来我们该怎么办呢？

首先，我们需要一个文本编辑器，比如 vi 或者类似于 IDEA 这样的集成编辑环境，把这段代码记录下来。文本编辑器，每个人都有不同的偏好，每个系统都有不同的偏好。一个软件工程师，可能需要很长时间，才能找到自己顺手的编辑器。就我自己而言，我使用了二十多年 vi 编辑器，直到这两年才发现 IDEA 的好。但是使用 IDEA 的时候，我还是会不自主地敲击 vi 的命令。不得不说，顺手，确实是一个很顽固、难改的行为习惯。

回到刚才的正题。有了文本编辑器，接下来，我们要把这段源代码编译成 Java 的字节码。编译器会帮助我们评估这段代码，看看有没有错误，有没有需要警示的地方。通常，我们使用 javac 命令行，或者通过集成编辑环境自动编译。

复制代码

```
$ javac HelloWorld.java
```

编译完成之后，我们要运行编译好的字节码，把程序的结果显示出来。在这里，我们一般使用 java 命令行，或者通过集成编辑环境来运行。

复制代码

```
$ java HelloWorld
```

最后一步，我们要观察运行的结果，检查一下是不是我们期望的结果。

复制代码

```
Hello, world!
```

如果让我去教授 Java 语言，教到这里，我会让同学们小小地庆祝一下：我们完成了 Java 语言的第一个程序。

万事开头难，完成 Java 语言的第一个小程序，尤其难！ 你要学习使用编辑器、使用编译器、使用运行环境。对于一个编程语言的初学者而言，这是迈入 Java 语言世界的第一步，也是很大的一步。这当然是巨大的收获，一个小小的庆祝当然是应得的也是值得的！

当然，会有同学试着改动这段代码，享受创造的乐趣。比如说，把“Hello, world!”改成“世界你好”或者“How are you?”。 这样一来，我们就还要经历编辑、编译、运行、观察这样的过程。

复制代码

```
class HowAreYou {
    public static void main(String[] args) {
        System.out.println("How are you?");
    }
}
```

毫不意外，对 Java 的了解更深之后，还会有同学继续修改代码，把 System.out 换成 System.err。然后，同样的过程还要再来一遍：编辑、编译、运行、观察。

其实，编辑、编译、运行、观察这四个步骤，就是我们学习一门新语言或者一项新特性的常规过程。如果你已经有多年的 Java 语言使用经验，想一想吧，你是怎么学习 JDK 7 的 try-with-resource 语句，又是怎么学习 JDK 8 的 Lambda 表达式的？是不是也是类似的过程？

也许，你已经习惯了这样的过程，并没有感觉得到有什么不妥当的地方。不过，如果我们看看 bash 脚本语言的处理，也许你会发现问题所在。

复制代码

```
bash $ echo Hello，World！
Hello, world!
bash $
```

显然，使用 bash 编写的“Hello, world!”要简单得多。你只需要在命令行输入代码，bash 就会自动检查语法，立即打印出结果；它不需要我们调用额外的编辑器、编译器以及解释器。当然，这并不是说 bash 不需要编译和运行过程。bash 只是把这些过程处理得自动化了，不再需要我们手动处理了。

拖后腿的学习效率

没有对比，就没有伤害。一般来说，不管是初学者还是熟练的程序员，使用 bash 都可以快速编写出“Hello, world!”，不到一分钟，我们就可以观察到结果了。但是如果使用 Java，一个初学者，也许需要半个小时甚至半天才能看到输出结果；而一个熟练的程序员，也需要几分钟甚至十几分钟才能完成整个过程。

这样的学习效率差异并不是无关紧要的。有来自学校的反馈表明，老师和学生放弃 Java 的最重要的原因，就是学习 Java 的门槛太高了，尤其是入门第一课。上面的这个小小的“Hello, world!”程序，需要极大的耐心，才能看到最后的结果。这当然影响了新的小伙伴们学习 Java 的热情。而且，老朋友们学习 Java 新技术的热情，以及深入学习现有技术的热情，也会受到了极大的阻碍。

JDK 17 发布的时候，我们经常可以看到这样的评论，“然而，我还是在使用 JDK 8”。确实，没有任何人，也没有任何理由责怪这样的用户。除非有着严格的自律和强烈的好奇心，没有人喜欢学习新东西，尤其是学习门槛比较高的时候。

如果需要半个小时，我们才能看一眼一个新特性的样子，重点是，这个新特性还不一定能对我们有帮助，那很可能我们就懒得去看了。或者，我们也就是看一眼介绍新特性的文档，很难有动手试一试的冲动。最后，我们对它的了解也就仅仅停留在“听过”或者“看过”的程度上，而不是进展到“练过”或者“用过”的程度。

那你试想一下，如果仅仅需要一分钟，我们就能看到一个新特性的样子呢？我想，在稍纵即逝的好奇心消逝之前，我们很有可能会尝试着动动手，看一看探索的成果。

实际上，学习新东西，及时的反馈能够给我们极大的激励，推动着我们深入地探索下去。那 Java 有没有办法，变得像 bash 那样，一分钟内就可以展示学习、探索的成果呢？

及时反馈的 JShell

办法是有的。JShell，也就是 Java 的交互式编程环境，是 Java 语言给出的其中一个答案。

JShell API 和工具提供了一种在 JShell 状态下交互式评估 Java 编程语言的声明、语句和表达式的方法。JShell 的状态包括不断发展的代码和执行状态。为了便于快速调查和编码，语句和表达式不需要出现在方法中，变量和方法也不需要出现在类中。

我们还是通过例子来理解上面的表述。

启动 JShell

JShell 的工具，是以 Java 命令的形式出现的。要想启动 JShell 的交互式编程环境，在控制台 shell 的命令行中输入 Java 的脚本语言命令 “ jshell ” 就可以了。

下面的这个例子，显示的就是启动 JShell 这个命令，以及 JShell 的反馈结果。

复制代码

```
$ jshell
|  Welcome to JShell -- Version 17
|  For an introduction type: /help intro

jshell>
```

我们可以看到，JShell 启动后，Java 的脚本语言就接管了原来的控制台。这时候，我们就可以使用 JShell 的各种功能了。

另外，JShell 的交互式编程环境，还有一个详细模式，能够提供更多的反馈结果。启用这个详尽模式的办法，就是使用“-v”这个命令行参数。我们使用 JShell 工具的主要目的之一，就是观察评估我们编写的代码片段。因此，我一般倾向于启用详细模式。这样，我就能够观察到更多的细节，有助于我更深入地了解我写的代码片段。

复制代码

```
$ jshell -v
|  Welcome to JShell -- Version 17
|  For an introduction type: /help intro
```

退出 JShell

JShell 启动后，就接管了原来的控制台。要想重新返回原来的控制台，我们就要退出 JShell。退出 JShell，需要使用 JShell 的命令行。

下面的这个例子，显示的就是怎么使用 JShell 的命令行，也就是“exit”，退出 java 的交互式编程环境。需要注意的是，JShell 的命令行是以斜杠开头的。

复制代码

```
jshell> /exit
|  Goodbye
```

JShell 的命令

除了退出命令，我们还可以使用帮助命令，来查看 JShell 支持的命令。比如，在 JDK 17 里，帮助命令的显示结果，其中的几行大致是下面这样：

复制代码

```
jshell> /help
|  Type a Java language expression, statement, or declaration.
|  Or type one of the following commands:
|  /list [<name or id>|-all|-start]
|    list the source you have typed
... snipped ...
|  /help [<command>|<subject>]
|    get information about using the jshell tool
... snipped ...
```

熟悉 JShell 支持的命令，能给我们带来很大的便利。限于篇幅，我们这里不讨论 JShell 支持的命令。但是，我希望你可以通过帮助命令，或者其他的文档，了解这些命令。它们可以帮助你更有效率地使用这个工具。

我相信你肯定会对帮助命令显示的第一句话非常感兴趣：输入 Java 语言的表达式、语句或者声明。下面我们就来重点了解一下这一部分。

立即执行的语句

首先，我们来看一看使用 JShell 来评估 Java 语言的语句。比如，我们可以使用 JShell 来完成打印“Hello, world!”这个例子。

复制代码

```
jshell> System.out.println("Hello, world!");
Hello, world!

jshell>
```

可以看到，一旦输入完成，JShell 立即就能返回执行的结果，而不再需要编辑器、编译器、解释器。

更方便的是，我们可以使用键盘的上移箭头，编辑上一次或者更前面的内容。如果我们想评估 System.out 其他的方法，比如不追加行的打印，我们编辑上一次的输入命令，把上面例子中的“println”换成“print”。就像下面这样就可以了。

复制代码

```
jshell> System.out.print("Hello, world!");
Hello, world!
jshell>
```

如果我们使用了错误的方法，或者不合法的语法，JShell 也能立即给出提示。

复制代码

```
jshell> System.out.println("Hello, world\!");
|  Error:
|  illegal escape character
|  System.out.println("Hello, world\!");
|                                    ^
```

JShell 的这种立即执行、及时反馈的特点，毫无疑问地，加快了我们学习和评估简单 Java 代码的速度，激励着我们去学习更多的东西，更深入的技能。

可覆盖的声明

另外，JShell 还有一个特别好用的功能。那就是，它支持变量的重复声明。JShell 是一个有状态的工具，这样我们就能够很方便地处理多个有关联的语句了。比如说，我们可以先试用一个变量来指代问候语，然后再使用标准输出打印出问候语。

复制代码

```
jshell> String greeting;
greeting ==> null
|  created variable greeting : String

jshell> String language = "English";
language ==> "English"
|  created variable language : String

jshell> greeting = switch (language) {
   ...>     case "English" -> "Hello";
   ...>     case "Spanish" -> "Hola";
   ...>     case "Chinese" -> "Nihao";
   ...>     default -> throw new RuntimeException("Unsupported language");
   ...> };
greeting ==> "Hello"
|  assigned to greeting : String

jshell> System.out.println(greeting);
Hello

jshell>
```

为了更方便地评估，你可以使用 JShell 运行变量的重复声明和类型变更。比如说，我们可以再次声明只带问候语的变量。

复制代码

```
jshell> String greeting = "Hola";
greeting ==> "Hola"
|  modified variable greeting : String
|    update overwrote variable greeting : String
```

或者，把这个变量声明成一个其他的类型，以便后续的代码使用。

复制代码

```
jshell> Integer greeting;
greeting ==> null
|  replaced variable greeting : Integer
```

```
4 |    update overwrote variable greeting : String
```

变量的声明可以重复，也可以转换类型，就像上一个声明并不存在一样。这样的特点和 Java 的可编译代码有所不同，在可编译的代码里，在一个变量的作用域内，这个变量的类型是不允许转变的，也不允许重复声明。

JShell 支持可覆盖的变量，主要是为了简化代码评估，解放我们的大脑。要不然，我们还得记住以前输入的、声明的变量，这可不是一个简单的任务。

也正是因为 JShell 支持可覆盖的变量，我们才能说 JShell 支持不断发展的代码，JShell 才能够更有效地处理多个关联的语句。

独白的表达式

前面我们说过，JShell 工具可以接受的输入包括 Java 语言的表达式、语句或者声明。刚才讨论了语句和声明的例子，现在我们来看看输入表达式是什么样子的。

我们知道，在 Java 程序里，语句是最小的可执行单位，表达式并不能单独存在。但是，JShell 却支持表达式的输入。比如说，输入“1+1”，JShell 会直接给出正确的结果。

复制代码

```
1 jshell> 1 + 1
2 $1 ==> 2
3 |  created scratch variable $1 : int
```

有了独立的表达式，我们就可以直接评估表达式，而不再需要把它附着在一个语句上了。毫无疑问，这简化了表达式的评估工作，使得我们可以更快地评估表达式。下面的例子，就可以用来探索字符串常量和字符串实例的联系和区别，而不需要复杂的解释性代码。

复制代码

```
1 jshell> "Hello, world" == "Hello, world"
2 $2 ==> true
3 |  created scratch variable $2 : boolean
4
5 jshell> "Hello, world" == new String("Hello, world")
6 $3 ==> false
7 |  created scratch variable $3 : boolean
```

总结

好，到这里，今天的课程就要结束了，我来做个小结。从前面的讨论中，我们了解了 JShell 的基本概念、它的表达形式以及编译的过程。

JShell 提供了一种在 JShell 状态下交互式评估 Java 编程语言的声明、语句和表达式的方法。JShell 的状态包括不断发展的代码和执行状态。为了便于快速调查和编码，语句和表达式不需要出现在方法中，变量和方法也不需要出现在类中。

JShell 的设计并不是为了取代 IDE。JShell 在处理简单的小逻辑，验证简单的小问题时，比 IDE 更有效率。如果我们能够在有限的几行代码中，把要验证的问题表达清楚，JShell 就能够快速地给出计算的结果。这一点，能够极大地提高我们的工作效率和学习热情。

但是，对于复杂逻辑的验证，使用 JShell 也许不是一个最优选择。这时候，也许使用 IDE 或者可编译的代码更合适。

我还拎出了几个技术要点，这些都可能在你的面试中出现。通过这一次学习，你应该能够：

了解 JShell 的基本概念，知道 JShell 有交互式工具，也有 API；

- 面试问题：你使用过 JShell 吗？

知道 JShell 能够接收 Java 编程语言的声明、语句和表达式，以及命令行；

- 面试问题：JShell 的代码和普通的可编译代码，有什么不一样？

这一次的讨论，主要是想让你认识到 JShell 能给我们带来的便利，知道简单的使用方法。这样，当后面我们想要讨论更多的话题时，你就可以使用 JShell 快速验证你的小问题、小想法。 要想掌握 JShell 更复杂的用法，请参考相关的文档或者材料。

思考题

在前面的讨论里，我们使用了一个例子，来说明 Java 处理字符串常量的方式。

复制代码

```
jshell> "Hello, world" == "Hello, world"
$2 ==> true
|  created scratch variable $2 : boolean
```

对于精通 Java 语言的同学，这个例子也许是直观的。但对部分同学来说，这个例子也许过于隐晦。过于隐晦的代码不是好的代码。同样地，过于隐晦的 JShell 片段也不是好的片段。

你有没有办法，让这个例子更容易理解？使用多个 JShell 片段，是不是更好理解？这就是我们今天的思考题。

欢迎你在留言区留言、讨论，分享你的阅读体验以及你对这个思考题的处理办法。

注：本文使用的完整的代码可以从GitHub下载，你可以通过修改GitHub上review template代码，完成这次的思考题。如果你想要分享你的修改或者想听听评审的意见，请提交一个 GitHub 的拉取请求（Pull Request），并把拉取请求的地址贴到留言里。这一小节的拉取请求代码，请放在实例匹配专用的代码评审目录下，建一个以你的名字命名的子目录，代码放到你专有的子目录里。比如，我的代码，就放在 jshell/review/xuelei 的目录下面。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 3 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 开篇词 | 拥抱Java新特性，像设计者一样工作和思考

下一篇 02 | 文字块：怎么编写所见即所得的字符串？

精选留言 (15)

写留言

旺仔double
2021-11-16

我把例子跟着打出来，比如那个switch，换行时会报错，这个问题大家遇到过没，我现在系统是win11，jdk11 ，求解答。

展开

作者回复: 这是一个很好的经验啊？JDK 11报的是语法错误吗？推荐使用JDK 17来学习这个专栏。这是一个switch的新用法。switch的问题，我们会在第5和第6讲里来说。

共 10 条评论 > 5

王位庆
2021-11-15

还停留在jdk8[捂脸]

展开

作者回复: 哈哈，老问题了。

共 3 条评论 > 3

星月
2021-11-17

范老师好，作为一名有些Java基础的图书策划编辑，我惦记新Java这个选题很久了哈哈，感谢您的这次课程。但我也会认真听课哒，力求为可能有的相关图书选题锦上添花！

作者回复: 出乎意料，居然有出版圈的朋友来。 谢谢来看这个专栏，希望可以帮到你。另外，现在图书编辑真是全能啊， 看得懂技术，够得着大势， 顾得上读者。

1

bigben
2021-11-16

有意思

展开

作者回复: 会越来越有意思的，这一讲就是个开胃的小菜。

1

都选C

2021-11-15

jshell> String firstString= "Hello, world";
firstString ==> "Hello, world"
| modified variable firstString : String
| update overwrote variable firstString : String…
展开

作者回复: 熟悉的ID啊，应该是老朋友了。

1

郭蕾

2021-11-19

老师，JShell 和我们经常说的 Shell 之间有关系吗？支持 Shell 的语法吗？

作者回复: JShell是Java的脚本语言，和Bash这样的操作系统的Shell是不一样的，语法的交集也很小。我们常说的Shell一般来说，指的是操作系统的脚本语言。

ABC

2021-11-18

JDK9发布的时候,学JShell刷过一遍文档,现在又看起来感觉很熟悉.用JShell的时候,初始化会有点卡,不清楚是不是机器配置问题.

之前在Windows下面编译OpenJDK的时候,遇到过一直Visual Studio不存在,后来发现需要安装在默认路径.那会想给OpenJDK提交个BUG,看了一下提交流程有点太复杂了,就暂时放…
展开

作者回复: 提交流程有很多中。最简单的，就是给jdk-dev@openjdk.java.net写点字邮件，把问题表述清楚就行了。 不过，如果你想自己填写bug，新手可能就会面临一点问题。

黄剑豪

2021-11-18

输入一下这段代码出错的规避方法：

```
jshell> greeting = switch (language) {
...> case "English" -> "Hello";
...> case "Spanish" -> "Hola";
...> case "Chinese" -> "Nihao"; …
```

展开

作者回复: 看起来是输入字符解析的bug。是的，复杂的输入使用编辑器更清晰些。

Jxin
2021-11-17

实现层面
1.仅展示的能力来看，感觉自己实现个jshll也不难。计算逻辑直接执行字符串就能拿到结果，特殊语法就做一层识别转换，变量就自己维护个上下文。

价值层面…

展开

作者回复: "需要着力推行在面向初学者的培训机构或学校的使用上"，如果教育和图书能够先行起来，对Java的发展会有巨大的推动。

注定非凡
2021-11-17

真是有趣的新特性，确实有很长一段时间没有关注JAVA的新特性了

作者回复: 快来一起学...

Geek_4c2e3e
2021-11-17

我也遇到了旺仔double说的问题，当输入完default -> throw new RuntimeException("Unsupported language");按下回车键后提示 错误:进行语法分析时已到达文件结尾，只能在不换行的情况下直接写上右大括号。我的环境是macOS 11.5.2，jdk是azul jdk 17.0.1 arm64。

展开

作者回复: 嗯，应该是个bug了。旺仔double稍后可能会给OpenJDK提交一个bug。

共 2 条评论 >

will

2021-11-16

老师，github连接在哪呢？

展开

作者回复: 每一讲的末尾，都有一个注。 说的就是GitHub的链接， 和PR请求提交的目录。

郡鸿

2021-11-16

这个用来验证一些小问题，计算结果，还真的是挺方便的！

作者回复: 是啊， 又快又好

Super~琪琪

2021-11-16

第三方包也支持吧？还是说需要配置下？
跟python的shell很像了。

作者回复: 支持，像普通的Java命令一样，使用"--class-path"把第三方包导入就好。当然，别忘了使用import。

3.141516

2021-11-15

话说 Java11 免费吗

展开

作者回复: OpenJDK都是免费的。 Oracle发布的版本，JDK 17也是免费的。是不是免费这个问题，有很多误解。你可以看看我在InfoQ上写的文章（https://xie.infoq.cn/article/a72ca12f97838e16ba2c91937）。 这个文章稍微有点老了，没有反应JDK 17之后的规则，不过大致的方向没有问题。